NASA TECHNICAL MEMORANDUM

NASA TM X-53342

July 1966

NASA TM X-53342

GPO PRICE	$
CFSTI PRICE(S)	$
Hard copy (HC)	4.00
Microfiche (MF)	1.00

ff 653 July 65

THE AMTRAN SAMPLER SYSTEM INSTRUCTION MANUAL (Revised)

By M.R. Albert, P.C. Clem, L.A. Flenker, J. Reinfelds, R.N. Seitz, and L.H. Wood

Research Projects Laboratory

NASA

George C. Marshall Space Flight Center, Huntsville, Alabama

JULY 1966 TM X-53342

THE AMTRAN SAMPLER SYSTEM INSTRUCTION MANUAL (Revised)

By

Albert, M. R.; Clem, P. C.; Flenker, L. A.;
Reinfelds, J.; Seitz, R. N.; and Wood, L. H.

George C. Marshall Space Flight Center
Huntsville, Alabama

ABSTRACT

3/2/8

Complete instructions are given for the efficient utilization of the AMTRAN Sampler software. AMTRAN (for Automatic Mathematical Translator) is an on-line remote terminal computer system. It permits the scientist and engineer to enter mathematical equations into the computer in their natural textbook format and to obtain immediate graphical displays of results. In addition, the professional programmer is provided with the capability to build high level operators which are indistinguishable from the basic intrinsic instruction set. The Sampler version of AMTRAN retains most of the software features of AMTRAN, but requires no hardware modification and can be used on any IBM 1620 computer with at least 40K storage.

NASA-GEORGE C. MARSHALL SPACE FLIGHT CENTER

TECHNICAL MEMORANDUM X-53342

THE AMTRAN SAMPLER SYSTEM INSTRUCTION MANUAL (Revised)

By

Flenker, L. A.; Reinfelds, J.;
Seitz, R. N. and Wood, L. H.
Research Projects Laboratory

and

Albert, M. R. and Clem, P. C.
Northrop Corporation - Huntsville, Alabama

RESEARCH PROJECTS LABORATORY

RESEARCH AND DEVELOPMENT OPERATIONS

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Concluded)

I. INTRODUCTION

I. INTRODUCTION

GENERAL COMMENT ON AMTRAN'S CAPABILITIES

AMTRAN is a multi-terminal conversational mode computer system designed to permit scientists and engineers to converse with the computer in the natural language of mathematics, entering mathematical equations into the computer according to textbook format and receiving immediate graphical and alphanumeric displays of results on a cathode ray scope and/or a typewriter. To permit the user to handle problems of a non-routine nature, AMTRAN also provides full programming capabilities. To give the system further flexibility, it is being designed to accept and run FORTRAN II programs.

A typical AMTRAN terminal consists of (1) a large function and operator keyboard with about 200 push buttons organized into operational groups, (2) a special typewriter which can type numerators and denominators, and (3) two cathode ray storage scopes, one for graphical and the other for alphanumeric display of results. A Polaroid camera provides a hard copy record of final graphical results. While AMTRAN will contain a real time FORTRAN II incremental compiler, its capabilities extend beyond FORTRAN in certain important ways. Two of the most important AMTRAN features are:

(1) AMTRAN provides automatic array arithmetic. Because the system can distinguish between a functional array and a constant, an operation defined on an array will automatically be performed on all its members. The user can, therefore, work with the normal entities of mathematics such as functions and constants.

(2) In addition to the usual programming capabilities, the user also has the ability to quickly and easily construct operators or macroinstructions which are indistinguishable from the repertoire of permanently programmed AMTRAN instructions. Furthermore, these operators can be nested or pyramided, to build powerful operators, which elicit, with one mnemonic or button push, a sequence of operations by the computer. The ease and speed with which these operators may be programmed makes it feasible to construct a library of macroinstructions and subroutines. This library is then stored on a disk and becomes available to all users.

AMTRAN'S SAMPLER

The AMTRAN Sampler system is a necessarily restricted version of AMTRAN and uses only card and typewriter for input and output. Because it is designed for a 40,000 digit IBM 1620, the Sampler does not require any special hardware, other than that stated under Hardware Requirements. The full AMTRAN system, however, takes advantage of more specialized hardware, such as disk storage, a high speed printer, cathode ray scope, and console keyboard.

Despite the limiting factor of the small size of available core storage area, the Sampler has many of the software capabilities of the extended AMTRAN language.

HARDWARE REQUIREMENTS

An IBM 1620 with the following minimum requirements is necessary for the Sampler's use:

1. 40 K Memory
2. Floating Pt. Hardware
3. Automatic divide
4. Indirect addressing
5. TNF, TNS, and MF instructions
6. IBM 1622 card reader/punch

Requirement 5 can be waived by slight modifications of the software. Requirement 2 can also be deferred but at a cost of about 3000 digits of the console program storage area.

LOADING THE SAMPLER SOFTWARE FOR A CHECKOUT OF THE SYSTEM

The Sampler's software consists of two parts:

1. A basic interpretive deck punched on consecutively numbered yellow cards, and
2. A register and console-program deck which is punched on white cards. The user is advised to duplicate both decks as soon as he receives them.

Instructions for loading the software and for setting the typewriter tabs and margins are given in Appendix A. After the Sampler program has been loaded into the computer, and the typewriter has typed

```
          ENTER PROGRAM
            1.
```

a check of the system can be made by typing the following test statement -

```
            1.  TYPEOUT 2 + 2.RS*
                      2 + 2 = 4
```

*Throughout this manual, the symbol RS (Release and Start) is used to refer to the following instruction:

> Press the RS button on the typewriter, or press the Release button and then the Start button on the 1620 console.

Unless the computer gives the above answer, something has gone drastically wrong. For example, the computer may not have the correct hardware configuration required by the Sampler. In the event of trouble, the user is advised to contact the authors of this manual.

PROGRAMMING IN AMTRAN

The Sampler contains a realtime interpreter of mathematical expressions, which permits the user to converse with the computer in the natural language of mathematics. To realize AMTRAN's full capabilities, the user needs to become familiar with a brief mnemonic language, simplified by the use of conventional mathematical labels. For example:

SIN	--	sine
COS	--	cosine
TAN	--	tangent
ABS	--	absolute value
LN	--	natural logarithm
SQ	--	square
SQRT	--	square root
PI	--	the constant (π = 3.1415927)
EXP	--	exponential

In addition to the mnemonic labels and their corresponding uses, certain systems procedures must be learned, such as putting a ".RS" at the end of a complete mathematical statement.

Once the mnemonic labels and system procedures are learned, mathematical expressions of any complexity can be written. Then, grouping these expressions together and adding input/output instructions and logic and branch operators, the user can build a sequence of statements which solve a particular problem. Such a grouping is called a program, which can be written to any length or complexity required.

II. THE AMTRAN LANGUAGE

II. THE AMTRAN LANGUAGE

MNEMONIC LABELS

In addition to the letters of the alphabet, the Arabic numerals, and the special characters mounted on an IBM 1620 console typewriter (- + * / . $ @ ,) the Sampler also uses variable length call codes with a maximum of 24 letters to call its various operations. Those fixed in the system are:

ALGEBRAIC OPERATORS

ABS	The absolute value
LEFT	Left Shift
RIGHT	Right Shift
SUM	Running Summation

RELATIONAL OPERATORS

AND	Used to link compound statements
EXCEEDS	Greater than ($>$)
IF	Compares two arithmetic expressions
LESSTHAN	Less than ($<$)

CONSTANTS

DEGREES	Converts degrees to radians
LG	The constant $\log_{10} e = .43429448$
PI	The constant $3.1415927 = \pi$

CORRECTION OPERATORS

DELETE	Deletes a program stored in core
EDIT	Used to change or delete a sentence
RESET	Resets the computer

EXPONENTIAL AND LOGARITHMIC OPERATORS

EXP	The exponential
LN	Natural logarithm
LOG	Logarith to the base 10

INPUT/OUTPUT OPERATORS

ENTER	Accepts a complete statement from the typewriter
INPUT	Accepts a single numerical typewriter entry
READ	Instruction given to read cards from the card reader
CARD	Gives a punched card deck of the register specified
DISPLAY	Gives a type written display of the register specified
DUMP	Gives a typed copy of the designated program
LIST	Causes a listing of the names of all console programs in the computer to be typed
PUNCHOUT	Punches a card deck of the specified programs
TYPE	Allows typewriter comment statements
W	Allows card comment statements

POWER OPERATORS

POWER	Raises a number to the specified power
SQ	Squares expressions
SQRT	Takes the square root of mathematical expressions

PROGRAM OPERATORS

EXECUTE	Reinstates the execution mode of operation
NAME	Assigns a mnemonic label to the given program
SUPPRESS	Suppresses execution of a program
CORE	Gives the number of locations remaining in the console program area

ARRAY DEFINERS

INTERVALS	The number of intervals between XMIN and XMAX
SUB	Subscript
XMIN	First abscissa value
XMAX	Last abscissa value

REGISTER OPERATORS

RESULTS	The temporary storage location of intermediate and final results
SETREG	Sets the number of function registers

TRANSFER OF CONTROL AND REPEAT OPERATORS

ALL	Used with DELETE, PUNCHOUT, and DUMP to designate all programs
END	Denotes the end of a program read from the card reader and causes computer control to come to the typewriter
EXIT	Causes a program exit
GO TO	Branch instruction
REPEAT	Repeats a designated program or expression an integer number of times
SWITCH	Used to represent Sense Switch 1 in an IF test

TRIGONOMETRIC OPERATORS

ARCTAN	Inverse tangent ($\tan^{-1}$)
COS	Cosine
SIN	Sine

For a detailed description of these constants and operators, including their exact uses, please consult Appendix B.

Either a space or a parenthesis must be placed between the mnemonic label and its argument. For a single variable or constant a space is necessary. For example:

```
1. SIN X .RS
2. ABS A .RS
3. EXP 12.3 .RS
4. TAN -17 .RS
5. COS PI
```

When the argument is a mathematical expression with more than one term it must be enclosed by parentheses.

```
1. COS(X+Y).RS
2. LOG(4X).RS
3. SQRT(B SQ -4AC).RS
4. COS(PI - 89.3 DEGREES).RS
5. ((A+B)DEGREES -X) SQ.RS
6. LN(6*10 POWER (A+X)).RS
```

If there is any question as to whether an expression requires parentheses, they should be used. GO TO is the only label that is not one continuous word; it must be separated by a space. Aside from this spacing is arbitrary. Additional spaces will be ignored by the computer and can be positioned to make expressions more readable. For comparison:

1.6×10^2 can be entered as

1.6 * 10 POWER 2 or

1.6*10 POWER 2.

$A + \frac{B}{C+F} - \text{SIN}\ (e^{\pi})$ can be written as

A+B/(C+F)-SIN(EXP PI) or

A + B/(C + F) - SIN(EXP PI).

All mnemonic labels precede the expressions on which they operate with the exception of DEGREES, SQ, and POWER, which follow their arguments.

III. ENTRY OF NUMBERS

III. ENTRY OF NUMBERS

DECIMALS AND INTEGERS

Numbers may be entered from the typewriter keyboard or from cards in any natural format. For example:

```
0
0.0
-1718.3026951
8 POWER 2
.008
.02000000
-123456789
6 POWER - 10.6
-26.3 POWER (A + 9.6671)
```

are all legitimate entries

The AMTRAN sampler accepts only 8 significant digits, and if more are entered, the ninth digit is used to round off the eighth digit to the nearest decimal place. Therefore, the number

1718.3026951 is interpreted by the computer as

1718.3027, and the number

123456789 is interpreted as

12345679, and the number

.008127197569 is interpreted as

.0081271976.

SCIENTIFIC NOTATION

Numbers with large positive and negative exponents could, as a matter of convenience, be entered in scientific notation--i.e., as a power of ten. For example,

.000000000015 could be entered as

15*10 POWER -12 or

1.5 * 10 POWER -11 or

.15 * 10 POWER -10.

All 3 entries are valid. The notation 15 * 10 POWER -12 is equivalent to 15 x 10^{-12} (scientific notation). Both the asterisk (a multiplication symbol) and the mnemonic POWER followed by a space or parenthesis are required when entering numbers in scientific notation. Examples:

1/100	becomes	1*10 POWER -2
	or	10 POWER -2
99999	becomes	9.9999 * 10 POWER 4
	or	99.999 * 10 POWER 3
12345678	becomes	1.2345678 * 10 POWER 8
16 x 10 (-12.3+6)	becomes	16 * 10 POWER (-12.3 + 6)

IV. FUNCTIONS, CONSTANTS, REGISTERS

IV. FUNCTIONS, CONSTANTS, REGISTERS

DEFINITIONS

This instruction manual uses the word constant to refer to an individual number which is stored in the machine, and the word function to refer to a list or array of numbers which constitute a numerical representation of a mathematical function. The word register is used to refer to a storage area which can hold either a constant or a list of numbers which represent a function.

STORAGE OF CONSTANTS

The Sampler possesses constant storage registers designated by the alphabetic letters A-R. These registers are assigned to represent programmed values. Example:

```
3 = A. RS
A-4 = B. RS
A+B = C. RS
```

It should be noted that a constant register will store only one number.

In addition to the A-R registers, there are three fixed constant registers, storing the frequently used numbers:

Mnemonic	Explanation of Mnemonic
PI	Stores 3.1415927
LG	Stores $\log_{10} e$ (.43429448)
DEGREES	Stores PI/180 and is used to convert degrees to radians

These constants can be called simply by typing or punching the appropriate mnemonic.

STORAGE OF FUNCTIONS

In the Sampler, functions (or arrays) are stored in registers. These registers are areas of computer core storage set aside to hold number arrays. There are eight such registers which are addressable by the alphabetic letters: S, T, U, V, W, X, Y, Z. In the 40 K version of AMTRAN, each register is capable of holding an array of 51 numbers, and in the 60 K version arrays of 101 numbers can be stored in each function register. Initially there are a total of 13 registers for the 40 K and 60 K versions. These can be referenced (or called) by the mnemonics REG 1, REG 2, ... REG 13, where REG 6 - REG 13 correspond to the registers S-Z. However, special care must be exercised when assigning arrays to REG 1 - REG 5, and therefore Appendix C should be consulted before storing arrays in these five registers.

To generate an array it is necessary to give the computer the following instruction:

RANGE OF N, i_1, i_2, i_3

where N = register name

i_1 = minimum value of array

i_2 = maximum value of array

i_3 = number of intervals between i_1 and i_2 (an interval size of i_3 requires i_3 + 1 numbers for its representation)

Examples:

(1) To generate an array over the range $0 \leq x \leq 1$, $\Delta x = .1$ enter:

```
RANGE OF X, 0, 1, 10.
```

10 intervals over the range from 0 to 1 makes $\Delta x = .1$ and gives an array in the X register equal to 0, .1, .2, .3, .4, .5, .6, .7, .8, 1. Once this array has been generated by a RANGE instruction, it can be henceforth called simply by entering "X". i.e.,

```
SIN X + COS X = W.RS
```

(2) To generate an array over the range $0 \leq Z \leq 10$, Z = .5, enter:

```
RANGE OF Z, 0, 10, 20.RS
```

This stores in the Z register the array:

0, .5, 1, 1.5, 2, 2.5, . . . 10.

(Note that an interval size of 10 generates an array of 11 numbers.)

To generate an array with more than 51 (40 K version) or 101 numbers (60 K version) a SETREG instruction can be declared. Please consult Appendix C for a detailed explanation of this procedure.

Subscripts are used to call an element in an array without calling the entire array. They are of the form:

X SUB 0 = first array value (X_0)

X SUB 1 = second array value (X_1)

⋮ ⋮ ⋮

X SUB N = last array value (X_n)

When the computer has completed the generation of X the typewriter returns carriage, tabulates, and then numbers for the next statement. To look at the contents of any register, turn computer console Sense Switch 2 on and enter DISPLAY followed by a register letter, such as DISPLAY X or DISPLAY T. The typewriter will then type the numerical representation of the function.

```
X SUB 0 =          X SUB 1 =
X SUB 2 =          X SUB 3 =
  : : :              : : :
X SUB N =
```

If Sense Switch 2 had been in the "off" position, only X SUB N would have been displayed.

V. ARITHMETIC OPERATIONS

V. ARITHMETIC OPERATIONS

INTRODUCTION

Since one of AMTRAN's goals is to enable the scientist or engineer to enter mathematical expressions in their natural textbook format, certain conventions are followed with respect to the arithmetic operations.

EQUAL SIGN

The equal sign in the Sampler is a replacement operator and can be thought of as a right pointing arrow. The numerical result of the left side is placed in the register or location to the right of the equal sign.

The expression: 1. A + 1 = A.RS is a valid equation. Suppose A has a certain initial value. Statement #1 causes the computer to go to the A register, add 1 to the value there, and store this result as the new value for A.

In a similar manner

```
X + Y = X.RS
```

is an allowable statement in AMTRAN.

In AMTRAN it is also possible to have a series of equalities in one statement. Example:

```
12.4 = A = B = C = D .RS
```

Constant registers A, B, C, D will all be filled with the value 12.4.

ARITHMETIC EXPRESSIONS IN GENERAL

All mathematical expressions are entered to the left of the "=" sign. Example:

```
A + SIN (45 DEGREES) = B .RS
```

Only a single letter variable or a subscript can be defined to the right of the equal sign; a number cannot be used.

The expression A = 2.RS is not a valid statement in AMTRAN. However,

```
      2 = A. RS
or    3 = X SUB 5. RS
```

are allowable.

All mathematical expressions are terminated by a ".RS". A series of expressions can be entered under one statement number by following each equation by a comma. Example:

```
1.  2 = A, A + 3 = B, A + B = C.RS
```

It is not necessary to declare a variable to the right of an equal sign. The expression `1. 2X + 3Y/10 .RS` is a valid statement in AMTRAN. When a transfer is not declared, the result of the expression remains in the floating accumulator (a register that moves up and down the stack of registers temporarily holding the results of computations) until the next arithmetic operation destroys it. Therefore, if an unstored computation is needed, it must be referenced immediately by the mnemonic RESULTS. Example:

```
1. 2X + 3Y/10.RS
2. RESULTS - Y SQ .RS
3. RESULTS/X = W.RS
```

In statement #2 RESULTS = 2X + 3Y/10, and
in statement #3 RESULTS = 2X + 3Y/10 - Y SQ.

A constant can be placed in any of the registers, constant or function. A functional array, however, can only be placed in a function register. Example:

```
1. X + 1 = A.RS
```

is an improper entry. A member of a functional array can be placed in a constant register. Example:

```
1. A = X SUB 0.RS
```

ADDITION AND SUBTRACTION

Additions and subtractions are performed exactly as they are in natural mathematics.

```
1. A + .1 = Y SUB 1 .RS
2. Y - X + .7542 = Z .RS
3. Y - (Z + 35). RS
```

MULTIPLICATION

Multiplication is an implied operation in AMTRAN and requires no special symbol. Example:

1. 2X + .3(Y + A) = Z.RS

2 is multiplied by X and then added to the quantity .3 multiplied by Y + A. The expressions:

1. Y X SUB 1 = Z.RS
2. X SUB 1 Y = Z.RS

are equivalent and are evaluated by the computer as Y multiplied by X SUB 1.

Because multiplication is an implied operation only single letters or subscripted variables can be used to represent constants and functions. For example, a constant designated as A2 would be interpreted by the computer to mean A is multiplied by 2.

Since one letter mnemonics are reserved for constant and function storage registers, when the computer receives a string of two or more letters from the keyboard, it compares this sequence with its table of mnemonics. If the computer fails to recognize a string of characters as a label, it interprets each letter as a storage register and carries out the implied multiplication. For example, ABCDE is interpreted as A*B*C*D*E*, because ABCDE is not a mnemonic lable.

When the user wishes to multiply the contents of two registers, he is responsible for insuring that a mnemonic is not accidentally generated.

For example, to multiply the contents of the C register by the contents of the O register and the contents of the S register, he should enter CSO rather than COS. In case of doubt, explicit multiplication symbols can be put between the letters. C*O*S will not be interpreted as calling the cosine.

Consider the constants intrinsic to the AMTRAN Sampler:

PI, XMAX, XMIN, INTERVALS, DEGREES, LG.

To multiply

$\pi \times 45^\circ \times .43429448 \times \mathrm{XMAX}$

it is necessary to use the explicit multiplication symbol or leave blanks between the constants.

Written in AMTRAN the above expression becomes

```
PI*45 DEGREES*LG*XMAX
```

or

```
PI 45 DEGREES LG XMAX
```

DIVISION

A diagonal slash mark "/" is AMTRAN's division symbol. The mathematical expression

$\frac{6(X+Y)}{B+C}$ is written in AMTRAN as

```
6(X + Y)/(B + C)
```

and

$\frac{10}{XY(A+B)}$ is entered as

```
10/(XY(A+B)).RS
```

If this instruction were written without the outside parentheses i.e.

10/XY(A+B).RS

the computer would interpret this as the expression

$$\frac{10}{X}Y(A+B).$$

VI. AUTOMATIC ARRAY ARITHMETIC

VI. AUTOMATIC ARRAY ARITHMETIC

The AMTRAN system has the ability to automatically distinguish between arrays of numbers, which constitute numerical representations of functions, and individual numbers, which represent constants or parameters. For example, in calculating SIN X, where X is an array of 50 numbers, the computer automatically calculates the sine of all 50 numbers in the array. On the other hand, to take the SIN A where A is an individual number, the computer automatically recognizes that A is an individual number and will take the sine only once. When a situation arises in which a constant must be added to, subtracted from, or multiplied by a functional array, the computer will automatically carry out the designated operation on every number in the array with the constant. Similarly, to divide one function Y by another function X, the computer will recognize that X and Y are functions and will divide every number in the Y array by the corresponding number in the X array.

VII. LOGICAL OPERATORS

VII. LOGIC OPERATORS

IF (for the comparison and testing of mathematical expressions)

In the evaluation of a mathematical expression, it is often necessary to determine whether the final result is greater than, less than, and/or equal to another expression or number, and depending on this information to carry out a special sequence of instructions or operations. Thus, AMTRAN provides an IF operator that allows mathematical expressions to be compared and tested.

The IF test is comprised of three clauses: an IF, a THEN, and an OTHERWISE. The IF clause specifies the expression to be tested, the relational conditions (less than, greater than, and/or equal) and the limiting term. The relational mnemonics are: LESSTHAN which is equivalent to the mathematical symbol $<$; EXCEEDS which is equivalent to $>$; "EXCEEDS=" which is equivalent to $\geq$; and "LESSTHAN=" which is equivalent to $\leq$.

In the IF test two functions (arrays) can be compared. Example:

```
1.  IF   X = Y, . . . .
2.  IF   X SQ - 17.3  LESSTHAN Y/X + SQRT Y, . . . .
3.  IF   X POWER 6  EXCEEDS = Z, . . . .
```

In statement 1, each value of the X array is compared with the corresponding value of the Y array, i.e., X SUB 0 is compared with Y SUB 0, X SUB 1 with Y SUB 1, ... X SUB N with Y SUB N. In statement 2, $(\text{X SUB 0})^2 - 17.3$ is compared with $\frac{\text{Y SUB 0}}{\text{X SUB 0}} + \sqrt{\text{Y SUB 0}}$, ... , $(\text{X SUB N})^2 - 17.3$ with $\frac{\text{Y SUB N}}{\text{X SUB N}} + \sqrt{\text{Y SUB N}}$. In a similar manner for statement 3, $(\text{X SUB 0})^6$ is compared with Z SUB 0, ... , and $(\text{X SUB N})^6$ with Z SUB N.

In the IF clause a function can be compared with a constant

IF X = 3.7, ...

IF EXP X + 22.3 EXCEEDS SIN (16 DEGREES), ...

IF SIN (5 DEGREES) = X, ...

Two constants can also be compared under the IF clause

IF B LESSTHAN .005, ...

IF A/B + 20 = SQRT 11.4, ...

(Note: In the IF clause an equal sign is not a replacement operator, and the expression $X + Y^2 - Z$ = Y POWER 3 + Z/100 is completely acceptable).

The THEN clause of an IF test tells the computer what to do when the IF clause is true. It can contain a branch statement:

```
IF A = B, THEN GO TO 3.RS
```

The "GO TO" mnemonic label causes control to be transferred to the statement number appearing after it (GO TO 3 tells the computer that the next statement to be executed is #3). When function arrays are involved in the IF clause, a GO TO declared in the THEN clause will result in a branching out of the IF. test. The statement IF A = X, THEN GO TO 2,will not test every member of the X array to determine if it is equal to A. It will test only those elements prior to its finding an X SUB J = A, and then it will branch out of the IF test and not return.

A mathematical expression can also be contained in the THEN clause. Example:

```
IF B = X, THEN X/B = Y, . . . .
```

An IF test can be embedded in the THEN clause.

```
IF X = Y, THEN IF Y = Z, THEN Z = 4, . . . . . .
```

The OTHERWISE clause tells the computer what to do when it encounters an invalid IF clause. As in the THEN clause, it can contain a branch statement, a mathematical equation, or an embedded IF test. When an array is involved in the IF clause, a GO TO declared in the OTHERWISE clause results in a branching out of the IF test. Example:

```
IF X = Y, THEN Z = Y, OTHERWISE GO TO 3.RS
```

When an X SUB K is encountered, such that X SUB K $\neq$ Y SUB K, control is transferred out of the IF test to the indicated statement as a result of the GO TO mnemonic in the OTHERWISE clause. Once control is transferred out of the IF test no return is made to test the remaining values in the X array.

The OTHERWISE clause can be omitted, in which case control falls through to the next statement after the IF test.

Both of the words THEN and OTHERWISE are dummy words and need not be stated except where clarity is desired. The commas separating the phrases, however, are necessary.

The IF test can be expanded through the use of the AND conjunction in both the THEN and OTHERWISE clauses

```
IF X = Y = Z, THEN B + 1 = X AND X/B = Y .RS
IF 3 = A, THEN A+1 = A,OTHERWISE 0 = A AND GO TO 3.RS
```

There is also an IF test that utilizes Sense Switch 1 on the IBM 1620 console. Example:

```
IF SWITCH 1, THEN . . . ., OTHERWISE . . . .
```

If Sense Switch 1 is on, control is transferred to the THEN clause, and if S.S.1 is in the "off" position, control goes to the OTHERWISE phrase. This option gives the user the opportunity to compare at execution time rather than at the time he programs.

REPEAT

It is often necessary to compute the result of a mathematical expression a specified number of times changing a parameter or array variable after each computation. The AMTRAN Sampler's REPEAT operator is designed to handle this logic problem. The instruction is:

```
REPEAT N, any mathematical expression .RS
```

(where N is the integer number of times the mathematical expression is to be repeated.)

N can be a number, a constant storage register (A-R), or a mathematical expression whose result is a single number. To illustrate:

```
ENTER PROGRAM
     1. REPEAT A + 3, INT EXP X = Y AND X + 1 = X.RS
```

(IF A + 3 happened to be a decimal number, it would automatically be rounded off to the nearest whole number.)

Any number of statements can be grouped together after the "REPEAT N", phrase provided that each one is separated by the mnemonic AND. In the above example, the expression INT EXP X = Y is computed A +3 times. After each computation of Y, the value of X is increased by 1.

In that section of the manual entitled "Console Programming" a method will be discussed whereby a lengthy series of mathematical statements and instructions can be indirectly declared and repeated N number of times by entering a single mnemonic label after the REPEAT phrase.

VIII. INTEGRATION AND DIFFERENTIATION

VIII. INTEGRATION AND DIFFERENTIATION

INTRODUCTION

The Sampler provides numerical integration and differentiation capabilities, both of which are AMTRAN subroutines (or console programs). The mnemonic INT calls integration and the mnemonic DERIV calls differentiation. The mnemonic labels are followed by a mathematical expression which must be enclosed in parentheses if it contains more than one term. In addition to numerical differentiation, the Sampler also has the ability to differentiate analytically. For the 40K version this procedure is available as a stand alone system.

INTEGRATION

To show how the numerical integration operator is used, it is best to begin with an example: To calculate

$$\int_0^1 \frac{(e^x + \text{SIN } x)}{(x+1)} \quad dx$$

enter

```
INT ((EXP X + SIN X) / (X + 1)) .RS
```

X must have been previously generated over the prescribed range with say 10 intervals. The numerical representation of the integral is then computed and the result stored in the accumulator. To obtain the result of this definite integral type

```
DISPLAY RESULTS .RS
```

For this instruction, Sense Switch 2 should be in the "off" position. The computer types

```
RESULTS SUB 10 = 1.4096106.
```

This is the definite integral of X from 0 to 1 with a step size of 0.1.

Note that no limits of integration were specified in the above integral. The computer automatically takes for its range of integration the range of X. Also note that no DX was used. Since the Sampler is designed primarily for functions of one independent variable, the variable of integration is implied and is generated from the abscissa values stored in the specified function register.

To store the above integral in a register (say Y) the user would type

```
RESULTS = Y.RS
```

Therefore, to obtain the correct definite integral, he would enter

```
DISPLAY Y.RS
```

The above integral, which we have obtained is actually the integral from 0 to X (in numerical form). As such, it differs from the indefinite integral F(X) only by a constant, F(0), where F(0) is the definite integral evaluated at the lower limit of the range of integration. Thus, the indefinite integral is given, for this example, by

F(X) = INT ((EXP X + SIN X)/(X+1)) + F(0) = Y + F(0)

This fact becomes important in carrying out a double integration.

F(X) = F(0) + INT ((EXP X + SIN X)/(X + 1)) + DF(0))

A factor to consider in using the Sampler's integration scheme is that of accuracy. The Sampler's fifth order integration formula provides seven to eight place accuracy if two conditions are met; 1) the range of integration must not be too large; 2) the user must be very careful in the neighborhood of a singularity. To illustrate these points, consider INT LN X over the range $.01 \leq x \leq 10.01$, using 100 intervals. This will give poor results for three reasons. First of all, the Sampler's integration scheme extrapolates the values of the integrand at two points to the right of the right-most tabulated value and at two points to the left of the left-most tabulated value. Thus, it extrapolates the values LN(-.09) and LN(-.19). Since LN(X) has a singularity at X = 0, and values of LN(-.09) and LN(-.19) are large and meaningless, and consequently, the integration procedure does not work well.

A second cause of trouble is the very rapid change in value of the integrand over the specified range. The integrand varies from - LN(100) to + LN(10). A third cause of difficulty resides in the fact that the higher derivatives are approximated in the integration procedure. Not only do these become very large (the fourth derivative is given by 24/X to the fourth power = 2,400,000,000), but they also grow progressively larger.

This invalidates the fundamental assumption upon which the integration is based, namely, that the first few terms of a power series expansion can be used to approximate the curve over a small interval. To illustrate this point, consider the contribution of the fourth-derivative term at the point X = .01 as used in the Sampler's integration scheme. Its contribution over the interval .1 to .2 is:

12/720 * 2,400,000,000 * .1 Power 5 = .433

The above three problems can be overcome by splitting up the range of integration into different parts, for instance, an integral from .01 to .101, an integral from .101 to 1.01, and an integral from 1.01 to 10.01 could be used to represent the integral over the range from .01 to 10.01.

The question logically arises as to how many intervals to use. There is no completely satisfactory answer to this question, but, after considering the above comments regarding singularities, a range of representation and a step size, should be tried, and then the same integration process repeated using a different number of intervals. If none of the numbers change except in the last decimal place, it can be assumed that a sufficient number of intervals have been employed. Note: Because of the integration scheme used, at least 5 intervals are required for the representation of an integral.

To increase accuracy in the integration process the correction of approximation errors was discussed relative to step size and singularities. However, another type of error exists which should also be considered. This is truncation error and is due to the dropping of digits in the running summation used in the integration over the range 0 to 1. The correct value of the integral from 0 to 1 is EXP (1) - EXP (0) = 1.7182818. Using 100 intervals, the value given by the computer is 1.7182811. Using 50 intervals, the computer gives 1.7182816. With 15 intervals, the computer gives 1.7182818, and with 10 intervals, the computer gives 1.718. Thus, roundoff errors decrease steadily, down to about 15 intervals, beyond which point the numerical representation of the function becomes inadequate.

NUMERICAL DIFFERENTIATION

Numerical differentiation, using forward differences, is available as an AMTRAN subroutine. It should be observed, however, that numerical differentiation is inherently less accurate than numerical integration because the methods of computation in each case are different. In numerical differentiation a small deviation between the actual and approximated curve over a given interval, although slight, may result in large differences of slopes. Errors of observation could also affect the computation of the derivative if such errors resulted in an alternation of the sign of closely spaced consecutive ordinates. Numerical integration schemes, on the other hand, are primarily smoothing procedures, summing over many intervals, and therefore, the associated errors may be small even though the interpolation polynomial is only a moderately good approximation of the function. In particular, then, numerical differentiation should be employed cautiously and avoided when data is empirical and subject to appreciable errors of observation.

To take the numerical derivative of

$$X^4 - \frac{x^3}{e^x} + 16\,x^2 - 145$$

the user would enter

```
DERIV( X POWER 4 - X POWER 3/EXP X + 16 X SQ - 145).RS
```

where X has been previously defined in a RANGE declaration.

Everything that has been said about integration applies even more stringently to differentiation. The Sampler's differentiation scheme has been derived by witchcraft. It has provided much better accuracy in various test cases than the formally correct differentiation scheme which corresponds to the Sampler's integration scheme. For this reason, the user should apply it with caution, using different ranges or interval numbers to check its accuracy.

SYMBOLIC DIFFERENTIATION

A special software feature is available as a complement to the AMTRAN system: symbolic differentiation. In the 40K version of the Sampler, it must be entered as a separate deck because of limited core space.

After the symbolic differentiation object deck has been read into the system, the computer is ready for the user to enter his equation or read in a console-programmed mathematical expression. Example: To differentiat

$$Y = X^2 e^x + 2X + 100$$

with respect to X the user would enter

```
Y = XXEXP X + 2X + 100, RS
```

The computer then types

```
DY/DX = 2XEXP X + XXEXP X + 2
```

after which a new equation can be immediately entered.

To obtain higher order derivatives the user enters the preceding derivative as the new expression for y. To find the second derivative of $Y = x^2 e^x + 2X + 100$ it is necessary to compute the first derivative, after which the user would enter DY/DX as Y:

```
Y = 2XEXP X + XXEXP X + 2,RS
```

The computer then types

```
DY/DX = 2EXP X + 4XEXP X + XXEXP X
```

which is the second derivative of Y = X^2 e^x + 2X + 100.

The symbolic differentiation program can differentiate (only with respect to X) any transcendental function of several variables written in terms of trigonometric, exponential, hyperbolic, logarithmic, or algebraic functions.

Expressions which are to be differentiated must contain only mathematical operators and operands, +, -, SIN, Y INT, etc. The only permissable exceptions are the period, and decimal pt.

IX. CONSOLE PROGRAMMING

IX. CONSOLE PROGRAMMING

INTRODUCTION

Up to this point in the manual, the Sampler has been presented as carrying out operations immediately as they are entered. Also, no provisions have been described whereby the operational sequences can be retained for future use. It is therefore necessary to introduce the programming capabilities of AMTRAN which give the system its power and flexibility and enable it to accommodate the logical syntactical language of mathematics.

Included in the programming capabilities provided in the Sampler is a common list of arithmetic operators that are used just as they would be in mathematics; trigonometric functions, hyperbolic and inverse trigonometric functions, exponential and logarithmic functions, powers, absolute value, left and right shift, running summation, generation of independent variables over a finite numerical range with a specified number of intervals, integral, derivative, lessthan, exceeds, and subscripting. In addition, there is an IF operator for testing expressions, editing capabilities for correcting programs, card & typewriter for input & output, and many other features. There also exist a group of instructions that tell the computer how to modify console programs during execution. For a complete list of instructions and operators in alphabetic order, the user should consult Appendix B.

Any sequence of these operators or instructions grouped together for a particular reason can be considered a "program." The term "console program" refers to any user-written program or to any program included in the Sampler's subroutine package (Section XII).

PROGRAMMING MECHANICS

After the Sampler deck has been read into the computer, control is automatically transferred to the typewriter which writes:

```
ENTER PROGRAM
    1.
```

The user then enters his expressions and operations. If a single expression does not fit on one line, it can be continued on the next line by returning the carriage of the typewriter. After a complete statement has been written and the ".RS" entered, the computer automatically renumbers for the next instruction.

Once the series of statements required for the solution of a particular problem have been completed, the set of instructions can be named. After this designation has occurred, the computer automatically readies itself to receive another program, and again types:

```
ENTER PROGRAM
    1.
```

In this way, programs can be written one right after the other. The present 40K system allows as many as 75 console programs to be written and entered at one time.

SPECIAL PROGRAMMING CAPABILITIES

NAME

Once a sequence of commands (or console program) has been written, it can be stored in the computer with the instruction NAME followed by a continuous mnemonic label of not more than 60 alphanumeric characters and with the stipulation that it does not duplicate any of those labels inherent in the system (see Appendix B) or any previously assigned to another program. If the name of a program duplicates that of another, the computer automatically types the error message:

ERROR-DUPLICATE LABEL, PLEASE REENTER STATEMENT.

A console program label, must begin with a letter, but can also include numbers in its declaration. The actual name of a program is completely a matter of preference and can be assigned any mnemonic label easily remembered.

To illustrate the use of the NAME instruction, consider the following example: A program to integrate the exponential of X over the range $0 \leq X \leq 5$ with 50 intervals ($\Delta X = 0.1$) and to retain the result in the system to use it again, would have the form:

```
1.  RANGE OF X, 0, 5, 50 .RS
2.  INT EXP X = Y.RS
3.  NAME INTEXP.RS
```

Henceforth, to call the integral of the exponential of X ($\int e^x dx$, $0 \leq x \leq 10$, $\Delta x = .1$) the mnemonic INTEXP is typed.

Programs can be nested inside each other, i.e., one program can call another one in the system. As many as 8 levels of nesting are permitted. For example, INTEXP calls INT which calls FORWARD. The mnemonic label of a program cannot occur before it has been declared in the NAME instruction; and therefore, a program cannot call itself.

In calling nested programs, special attention should be given to changes in variables within these levels of program nesting. To illustrate: Suppose that the following program is written

```
1.  RANGE OF Y, 10, 20, 100 .RS
2.  RANGE OF X, 0, 10, 100 .RS
3.  INT EXP X + Y = Z.RS
```

It is remembered that statement 2 and INT EXP X of 3. have already been programmed in the console program INTEXP. Therefore, statement 2. is deleted and the third statement changed to read 3. INTEXP + Y = Z.RS

But, in the program INTEXP, INT EXP X = Y. Therefore, the Y in statement 3. will not be $10 \leq Y \leq 20$, $\Delta y = .1$, but rather the Y in program INTEXP.

The above example illustrates the fact that all of the variables (A-Z) used in the Sample are global. This is to say, that they are used and called by the same name in all subroutines.

READ

The READ instruction transfers control from the typewriter to the card reader. This operator can be typed at any time that the user chooses to enter a console program or programs punched on cards.

PUNCHOUT

Hard copy of a console program can be obtained by typing PUNCHOUT, followed by its name, causing the program to be punched out on cards in source language. Consider the program INTEXP. The instruction for outputting this on cards would be:

```
1.  PUNCHOUT INTEXP.RS
```

By making a compound statement

```
1.  PUNCHOUT ALL.RS
```

a hard copy (punched card deck) can be obtained of all the console programs in the system at the time the instruction is given. Each punched program carries its own mnemonic label. This gives a separable form of console program output so that a card file library of console programs can be established. To reenter any combination of these programs into the system at some future time, the user would place the desired deck in the card reader and type

```
READ.RS
```

This procedure would reinsert into the computer the console program set which has been loaded from the reader.

DUMP

DUMP is used in the same manner as PUNCHOUT, except that it gives a typewriter printout of a program. DUMP ALL will elicit a typeout of all existing programs in core.

LIST

To determine the selection of console programs within the computer at any time, a typeout of the console programs' mnemonic labels can be obtained by entering:

```
1. LIST. RS
```

Consider the hypothetical case in which a program is written requiring the integral of a function. To determine if the program which allows integration is in the system type:

```
1. LIST.RS
```

A list of console program labels is displayed for the user's inspection. If the mnemonic INT is found, the system contains the integration subroutine and will properly interpret the INT mnemonic label. It must also be ascertained that nested programs called for by the _LIST_ declared programs are in the computer. For example, INT calls the program FORWARD. Therefore, it is not enough that INT be in the _LIST_ declaration, but that FORWARD also be there.

DELETE

A console program in the system can be eliminated at any time with the instruction _DELETE_ followed by the name of the program. By constructing a compound statement,

1. DELETE ALL.RS

all of the console programs in the system at the time can be destroyed.

Deleting a program frees the mnemonic call label and the associated core area, which can therefore be used to store another program.

If Sense Switch 2 is in the "on" position when a program is deleted, the computer will type out the number of core locations left in the console program storage area.

EDIT

Unlike the DELETE operator, which destroys an entire program, the EDIT instruction is used to change or delete statements in programs that have already been written.

To change or delete a statement, the EDIT mnemonic is entered, followed by the name of the program and the statement number.

Suppose that, after having written a program, called EXAMPLE, it is decided for some reason that statement 3 should be changed. The instruction is:

```
EDIT EXAMPLE 3. RS
```

The computer deletes this statement in the program, types a 3. and then waits for the new or corrected statement to be entered. To delete the statement without entering a replacement, a ".RS" is typed.

The EDIT operator can also be used to add statements to a program by converting a single statement into a series of statements, each separated by a comma. For example: To insert a statement between statements 3 and 4 of program EXAMPLE, rewrite statement 3 under the EDIT instruction terminating it with a comma instead of a period. Following the comma, insert the statement or statements required between statements 3 and 4.

(the original program EXAMPLE)

```
ENTER PROGRAM

    1.  RANGE OF X, 1, 2, 10.RS

    2.  RANGE OF Y, 2.2, 3.2, 10.RS

    3.  X SQ + Y SQ = Z .RS

    4.  SQRT Z = T .RS

    5.  NAME EXAMPLE .RS


ENTER PROGRAM

    1.  EDIT EXAMPLE 3 .RS
```

(statement 3 with Z SQ = Z added)

```
3.  X SQ + Y SQ = Z, Z SQ = Z. RS
```

The same insertion could have been made by rewriting statement 4, preceeding it with Z SQ = Z.

```
EDIT EXAMPLE 4 .RS

    4.  Z SQ = Z, SQRT Z = T .RS
```

RESET

The RESET instruction is mentioned here because it allows the destruction of a program before it has been completely named. If, after having written, say, 6 instructions for a program, the users finds that he is completely wrong in his logic or approach, he can type

```
RESET .RS
```

as his 7th statement. This completely wipes out what has been written since the last ENTER PROGRAM declaration. It does not, however, wipe out any data from constant registers or function arrays.

CORE

By entering CORE.RS a statement as to the number of unused core locations remaining in the console program storage area can be obtained. As a reply to this instruction the computer types

```
XXXXX OPERATIONS LEFT.
```

Where XXXXX is a 5 digit number. For each operation left, one operator or register can be entered. Example:

SIN X requires 2 locations

A + B requires 3 locations

DISPLAY T requires 2 locations

This label is particularly useful in determining the remaining core space available for console programs.

EXIT

The EXIT mnemonic causes computer control to be transferred out of the program in which it appears. To illustrate consider the following program:

```
1.  3.45 = A. RS

2.  RANGE OF X, 7, 8, 10. RS

3.  X SQ + X/(A SQ) + 1/A = Y.RS

4.  DISPLAY Y.RS

5.  IF A LESSTHAN = 4, THEN A + .01 = A AND GO TO 3,
    OTHERWISE EXIT.RS

6.  NAME POLYNOMIAL .RS
```

In the program POLYNOMIAL, Y is computed for $3.45 \leq A \leq 4$, $\Delta A = .01$. Statement 5 tests A to determine if it is in this range. When A becomes greater than 4, the EXIT mnemonic in the OTHERWISE clause will cause a branching out of the program POLYNOMIAL. Once the branch has occurred the typewriter types:

```
ENTER PROGRAM
     1.
```

and waits for a new series of statements to be entered.

The EXIT operator can also be used in the THEN clause of an IF test. Where functional arrays are concerned in the IF test, it should be remembered, that as soon as an EXIT is encountered in the THEN clause (for a valid IF) or in the OTHERWISE (for an invalid IF), computer control branches out of the program and does not return to test the remaining array values.

The EXIT mnemonic can only be employed when the computer is in the suppressed mode of operation (consult the next section under

Console Programming entitled "Modes of Operation" for a detailed description of this mode).

REPEAT

As previously explained in the section on Logic Operators (VII) the REPEAT mnemonic can be used to repeat a series of directly declared mathematical statements (the maximum number of times a statement can be repeated is 999). It is also possible to repeat a console program a specified number of times, thus indirectly declaring a sequence of instructions and mathematical expressions. The instruction given is "REPEAT N", followed by the program's mnemonic name. (N is the number of times the program is to be repeated). To repeat the program EXAMPLE 10 times enter:

1. REPEAT 10, EXAMPLE .RS

Note: The REPEAT mnemonic can only be entered in the suppressed mode.

TRANSFER

The TRANSFER operator is used whenever it is necessary to move symbols about in a program or between different programs. It is employed by the instruction, TRANSFER J TO K, where J and K are variable program line positions. To transfer symbols within a program $J = j_1 . j_2$ and $K = k_1 . k_2$ such that j_1 = statement number = k_1 (j_1 and k_1 are not necessarily equal). and j_2=symbol position = k_2 (j_2 and k_2 are not necessarily equal). A symbol position is required for each of the following: a mnemonic label, comma, parentheses, period, mathematical operator, register, and a single numerical digit. For Example:

```
2.  SIN (3.2 DEGREES) = A.RS
```

The SIN occupies the first symbol position, and A the ninth symbol position. To move SIN, the instruction would be:

```
TRANSFER 2.1 to . . . .
```

and to transfer DEGREES

```
TRANSFER 2.6 to . . . .
```

To complete the discussion of the TRANSFER instruction consider the following two statements:

```
2.  SIN (3.2 DEGREES) = A.RS
3.  SQRT (ATAN (10 DEGREES)-A) = B.RS
```

To transfer SIN (sine) in statement 2 to ATAN (arc tangent) in statement 3 the entry would be:

```
TRANSFER 2.1 to 3.3.RS
```

After the transfer has occurred, statement 2 remains unchanged, but statement 3 now reads:

```
3.  SQRT (SIN(10 DEGREES)-A) = B.RS
```

This illustrates the fact that when a transfer occurs, the line and symbol position to the left of the TO retains that symbol it held prior to the TRANSFER instruction. Only the symbol declared on the right side of the TO is replaced.

It should be noted that a transfer can only occur on a one to one basis. For example, the 10 in statement 3 couldn't be transferred to the place occupied by the 3.2 in statement 2 and vice versa, because the 10 takes two symbol positions, while 3.2 requires three symbol positions.

To move a symbol between programs the mnemonic name of the two programs must be introduced in the TRANSFER instruction. In the preceeding example, if statement 2 had been in program TEST and statement 3 had been in program TRIG, then to replace the ATAN with the SIN, the instruction would have been:

```
1.  TRANSFER TEST 2.1 TO TRIG 3.3.RS
```

The TRANSFER operator, like the EXIT mnemonic, can only be used in the suppressed mode.

MODES OF OPERATION

Depending on the operating mode of the computer, the user's commands are stored with or without simultaneous execution. Two modes of operation are available in the Sampler, an execution mode and a suppressed mode.

The execution mode is the normal state of operation for the computer. As each statement is entered, it is immediately executed. There are seven mnemonic restrictions in this mode: the GO TO, EXIT, REPEAT, TRANSFER, ENTER, ENTRY, and INPUT instructions cannot be used.

The suppressed mode allows the entry and storage of a program into the computer without the simultaneous execution of its statements. To achieve this state, type

```
SUPPRESS. RS
```

to which the computer replies

```
ENTER PROGRAM-SUPPRESSED
    1.
```

This mode allows the user to write a complete program without taking the time between instructions for execution. Having written and named the program, the user can choose to execute it simply by entering its label followed by a ".RS". Once a program written in the suppressed mode is named, the computer automatically drops out of this mode and waits for a new program to be entered in the execution mode. If at any point before the NAME instruction, the user wants to transfer to the execution mode, he types EXECUTE .RS.

GENERAL PURPOSE PROGRAMS

Up to now, the user has been inputting all of his information into the computer as he writes a program. It is possible, however, to write a general program in which input values and mathematical expressions are entered only at execution time. This allows the construction of general-purpose macro-instructions which can be used with arbitrary operands. The general-purpose programming is made possible by two AMTRAN instructions: INPUT and ENTER.

To illustrate the use of the INPUT mnemonic, consider the example INTEXP. To make this a general purpose program where any range and step size of X could be used, the user could have written:

```
1. RANGE OF X INPUT, INPUT, INPUT.RS
2. INT EXP X = Y.RS
3. NAME INTEXP. RS
```

The mnemonic INPUT causes the computer to branch out of the console program INTEXP and to come to the typewriter for an entry. Here, the computer waits for the minimum value of X, the maximum value of X, and the number of intervals in the range of X to be entered. After these entries have been made, a ".RS" or a ",RS" must be typed to return control back to the console program.

The INPUT operator can be used for any constant or mathematical expression that changes with each execution of a program. A mathematical expression of more than one term must be enclosed in parentheses when entered. To illustrate consider the following 2 examples:

To input a constant:

```
ENTER PROGRAM-SUPPRESSED
    1. SIN INPUT = A. RS
```

```
2.  1/A = K.RS
3.  DISPLAY K.RS
4.  NAME CSC.RS
```

To call (or execute) the program CSC the entry would be:

```
1.  CSC RS
```

(Note: The computer must be in the execute mode, and no period is declared before the RS). The computer will then begin to execute the program CSC until the INPUT instruction is encountered. Then computer control is transferred to the typewriter to accept the input value (say .171). To return control back into the program a ".RS" must be typed after the entry.

As an alternate method of executing the program, the input value can be declared immediately after the program label. Example:

```
1.  CSC .171 .RS
```

If a program contains several input values, they can all be entered after the program name provided that each entry is followed by a comma. Consider the program:

```
ENTER PROGRAM-SUPPRESSED
    1.  SIN INPUT = A.RS
    2.  COS INPUT = B.RS
    3.  TAN INPUT = C.RS
```

4. 1/A = K.RS

5. 1/B = L.RS

6. 1/C = M.RS

7. DISPLAY K, DISPLAY L, DISPLAY M.RS

8. NAME TRIG.RS

To execute this program and input the constants (.171, .333, 1.956) type:

1. TRIG .171, .333, 1.956.RS

To input a mathematical expression:

If in the program CSC the entry had been 13 DEGREES, this is considered a mathematical expression and must be enclosed in parentheses. For Example:

1. CSC (13 DEGREES) .RS

If "SIN INPUT = X" in program CSC, then a possible input might be:

1. CSC ((Y SQ + ABS Y)DEGREES).RS (where Y has been previously defined)

The ENTER mnemonic, the AMTRAN Sampler's other general purpose program feature, permits the entry of a complete mathematical expression or "code string". The mnemonic ENTRY is used in conjunction with ENTER, to execute an ENTER declared statement. Example:

```
ENTER PROGRAM-SUPPRESSED

    1.  RANGE OF X, 1, 2, 20.RS

    2.  ENTER. RS

    3.  REPEAT 5, ENTRY AND A + 1 = A.RS

    4.  DISPLAY Y.RS

    5.  NAME EQUATION.RS
```

The ENTRY in statement 3 causes the execution of the expression entered in statement 2, and repeats the computation, changing A five times. If ENTRY had been an INPUT the expression would have to be entered manually from the typewriter before each computation. Herein lies the difference between the ENTER and INPUT. An expression can be typed once under the ENTER instruction, and then referenced and manipulated by the ENTRY mnemonic without having to type that expression each time it is used. An ENTRY always refers to the ENTER expression just preceeding it. However, when entering an expression or constant under INPUT, the computer requires a typed value each time the mnemonic label INPUT is encountered.

To execute the program EQUATION and input the expression

$$Y = X^2 - 7.3\ X + 99$$

the user would type:

```
1.  EQUATION RS
```

The computer then proceeds to execute the program until it comes to the mnemonic ENTER. Then computer control comes to the typewriter to accept the mathematical expression

X SQ - 7.3 X + 99 = Y.RS

(The ".RS" returns control back to the program)

An alternate method of entry would be to declare the statement immediately after the program name. Example:

1. EQUATION X SQ - 7.3 X + 99 = Y.RS

Note: The ENTER mnemonic can only be declared in the suppressed mode.

PROGRAMMING LEVELS

The concept of programming levels is of special importance when considering the nesting of programs. The nesting of programs provides the ability for the building of high level operators which greatly extend the intrinsic AMTRAN Sampler instruction set. This ability encourages block programming in which segments of a program are written and checked a block at a time and then combined to create more complex programs.

The INPUT operator embedded in a nested program gives a graphic illustration of levels of program nesting. Consider a problem in optics associated with rectangular slit diffraction patterns: the Fresnel

Integrals which give the x and y coordinates along the Cornu spiral. The following are two programs which generate the FRESNEL integral:

```
ENTER PROGRAM-SUPPRESSED

        1.  INT SIN (PI INPUT SQ/2).RS

        2.  NAME PRESNELSINE .RS
```

and

```
ENTER PROGRAM-SUPPRESSED

        1.  INT COS (PI INPUT SQ/2) .RS

        2.  NAME FRESNELCOSINE.RS
```

The FRESNELSINE and FRESNELCOSINE programs are considered here to be mainline or on the 1st level of nesting programs. They both call INT which is therefore on the 2nd level of nesting. INT calls FORWARD, which becomes a 3rd level of nesting.

Consider a third program which calls the previous two.

```
ENTER PROGRAM-SUPPRESSED

        1.  RANGE OF T, INPUT, INPUT, INPUT.RS

        2.  FRESNELCOSINE T = X.RS

        3.  FRESNELSINE T = Y.RS

        4.  DISPLAY X, DISPLAY Y.RS

        5.  NAME CORNU.RS
```

In the program CORNU, FRESNELSINE and FRESNELCOSINE become the second level of nested programs, and CORNU the first.

When the FRESNELSINE is a mainline program, the INPUT operator causes the computer to come to the typewriter and accept an entry. This is also true in the FRESNELCOSINE program. Now consider the program CORNU. The INPUT operators in statement 1 cause the computer to come to the typewriter and accept a maximum value, minimum value, and interval number for T. In statements 2 and 3, the computer branches to the declared programs. This time, when the INPUT operator is encountered it does not cause the computer to branch to the typewriter, but rather up into the CORNU program where the T array is taken as the INPUT.

Programming level changes have occurred in the previous example. Program nesting is equivalent to program levels, but the latter term is used to clarify the movement of computer control as it branches from console program to console program. Every time the INPUT operator is used, it causes the computer to move up one level (or program). Therefore an INPUT declared in an embedded program, will cause the computer to move up one level to the program in which it is nested and look for its argument. This is true for all the possible 8 levels of program nesting.

In nested programs which require input, a variable or an INPUT must be declared after the subroutine label. To illustrate:

In CORNU both FRESNELSINE and FRESNELCOSINE declare a T after their mnemonics. If the user had not wanted to draw the T value from the CORNU program, but rather from the typewriter, he would have entered INPUT instead of T. That is,

```
FRESNELSINE      INPUT.RS

FRESNELCOSINE    INPUT.RS
```

X. INPUT/OUTPUT

INTRODUCTION

Two types of input and output are available with the AMTRAN Sampler: card and typewriter. Both require that the basic interpretive Sampler card deck be first read into the computer. After this is done, the typewriter will type

```
        ENTER PROGRAM
            1.
```

signaling that the system is ready to accept mathematical equations or instructions. A standard package of console programs (Section XII) can be read into the computer immediately after the basic interpretive Sampler card deck. These are:

FORWARD	Computes the forward difference.
RANGE OF	Generates an array over a specified interval and step size.
INT	Numerically integrates the expression following it.
DERIV	Takes the numerical derivative.
TYPEOUT	Gives a typewriter display of equation and results.
SET	Inputs data in any format.

For a complete description of these subroutines, see Section XII entitled "Programs in the AMTRAN Sampler Subroutine Package:.

INPUT/OUTPUT OF CONSOLE PROGRAMS

To input a console program punched on cards, the instruction

1. READ .RS

must be entered at the typewriter. Any number of console programs can then be read into the computer one immediately following the other. If the programs are to be executed as they are read in, a card with the mnemonic EXECUTE punched in columns 7 - 14 must be placed at the beginning of the program deck, otherwise, all programs are read in the suppressed mode. Programs with a GO TO, ENTER, TRANSFER, ENTRY, REPEAT, EXIT, or an INPUT cannot be entered under the EXECUTE card heading. At the end of the deck of programs to be read in there must be a card with the label END punched in columns 7 - 9. This card returns control to the typewriter. In all card input statement numbers are restricted to columns 1 - 6 and statements or data to columns 7 - 80.

Card output of a console program can be obtained (1) with the PUNCHOUT instruction, followed by the name of the program or (2) with PUNCHOUT ALL which gives a card source deck of all the console programs in the system. These cards may then be passed through an IBM keypunch for labeling purposes.

Typewriter input of a console program is anything that the user enters from the keyboard between the typewriter message:

ENTER PROGRAM

and the NAME instruction. Typewriter output of a console program is obtained by a DUMP instruction followed by the name of the program.

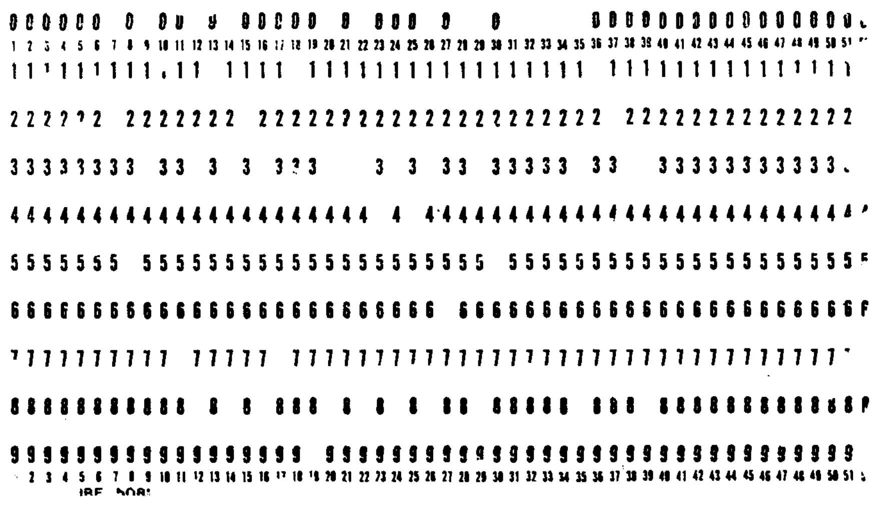
INPUT AND OUTPUT OF DATA

For the input of data on cards, there is a special console program named SET contained in the Sampler subroutine package. The user needs to enter, beginning in column 7 of the card:

1. The name of the subroutine SET
2. A space
3. The number of data points followed by a comma
4. The data points one after the other and each separated by a comma
5. A period after the last data point

To illustrate, consider the following example: To enter an array of seven values into the X register the user types "READ .RS" and then reads in the SET program. All cards are automatically read in the suppressed mode unless otherwise specified. To execute SET and input data on cards an EXECUTE card is placed after the last card in subroutine Set followed by a card in this format:

SET 7,1,2.719,3,4.4,60,50000,123.



Note: Input values can be in any format. Subroutine SET reads all input values into the X array. Therefore, to reference the data points read in the user should address the X array (X SUB 0 will contain the first input value, X SUB 1 the second, and etc.).

To input data while at the typewriter subroutine SET is called as a nested console program. If the data values are to be entered from the typewriter at execution time the mnemonic INPUT is declared after SET. For example:

```
ENTER PROGRAM-SUPPRESSED
      1.  EXP SET INPUT = Z.RS
      2.  SQRT Z - 63.2 Z POWER 10. RS
```

```
            3.  CARD RESULTS .RS
            4.  NAME DATAEXAMPLE .RS
```

To execute the program DATAEXAMPLE and input that set of values given in the previous example on card input, the instruction would be:

```
        ENTER PROGRAM
            1.  DATAEXAMPLE 7, 1, 2.719, 3, 4.4, 60, 50000, 123 .RS
```

The first entry (a 7 in this example) declared after the program name must always be the number of data points.

Statement 1 in program DATAEXAMPLE causes the exponential of the input values to be stored in the Z register. Statement 3. gives the result of statement 2 on cards. If the expression had been

```
            2.  SQRT Z - 63.2 Z POWER 10 = Y.RS
```

then statement 3 could have been entered as:

```
            3.  CARD Y. RS
```

The instruction CARD can be used to obtain the output of any register (a constant or a function) on cards. In the case of a function, two array values are punched per card.

The corresponding mnemonic for typewriter output of a constant or array is DISPLAY. For typewriter output of statement 2., the instruction would be

```
            3.  DISPLAY RESULTS.RS
```

or, if the expression had been

```
2.  SQRT Z - 63.2 Z POWER 10 = Y.RS
```

statement 3. could have been:

```
3.  DISPLAY Y.RS
```

The typewriter output of a function register can be further controlled with console Sense Switch 2(S.S.2). If S.S.2 is in the "on" position, all values of the array will be typed, and if in the "off" position, only the last value of the array will be given.

Output is displayed as:

```
X SUB 0 =                X SUB 1 =

X SUB 2 =                X SUB 3 =
```

FORMAT CONTROL

Since the SET subroutine permits data to be entered in any mixed format, format statements are not required in AMTRAN for the input of values. The comma separating each number tells the computer that a new value is being entered, and a period after the last value designates the end of a sequence. All card and typewriter output of data is automatically given in floating point notation with 8 significant digits punched or typed 2 data points per card or line.

To output values in a different format (as, for example, in obtaining tabular results), format control statements are provided in the Sampler. The symbols @ (at sign) and $ (dollar sign), used in conjunction with the mnemonic TYPE or PUNCH, designate format control. Three different types of formated numerical output are available in the Sampler.

The first type is called a "left justified fixed field format" and is designated by an @ sign. Consider the constant register A such that A holds the value .00000016790924. To obtain the contents of A in a left justified fixed field format, A is preceded by an @. The instruction to the computer would be:

```
TYPE A = @A.RS
```

The computer's typewritten response is:

```
A = .16790924 * 10 POWER(-5)
```

The TYPE mnemonic is used to print out characters and statements as comments or documentary (see the next section on "Comment Statements". Therefore, TYPE A =   , causes the computer to type the two characters "A =   ." Note: The size of the fixed field under the @ formating allows the output of two values per line of type.

"Decimal justified fixed field" formating is provided by the $ sign. This type of formating fixes the decimal point and distributes the numbers around it accordingly. If a series of values are to be outputted, this format lines up all the decimal pts. and exponents one under the other. For example: Consider the following array values contained in the X register:

X_0 = .0000016790924

X_1 = .0000010548213

X_2 = .00000041004291

X_3 = .000000089997612

To output these values in a decimal justified format type:

```
TYPE X = $X .RS
```

The computer then responds by typing:

```
X =    .16790924      *    10 POWER(-5)
X =    .10548213      *    10 POWER(-5)
X =    .41004291      *    10 POWER(-6)
X =    .899997612     *    10 POWER(-7)
```

Note: The size of the fixed field under the single $ format control permits one output value per time of type.

"Variable" formating is provided by the use of more than one $, where each dollar sign defines a decimal place. As an example, consider the constant register B where B holds the value 22.779. Depending on the variable format desired, the user would instruct the computer to:

```
TYPE B = $$B.RS
```

to which the computer responds:

```
B = 23
```

or

```
TYPE B = $$$.B .RS
```

to which the computer responds:

```
B = 22.8
```

or

```
TYPE B = $$$.$$B.RS
```

to which the computer responds:

```
B = 22.779
```

Note: An extra place should always be provided for the sign of a number (either plus or minus). Also, the register name declares a decimal place. The number of output values per line of type depends on the size of each field.

If formated output is wanted on cards instead of from the typewriter the mnemonic TYPE in the above examples should be replaced by the mnemonic PUNCH.

COMMENT STATEMENTS

There are three possible comment statements available in the AMTRAN Sampler, two for card, and one for the typewriter. The TYPE instruction followed by a message is used for typewriter comments. During execution, the statement following a TYPE is printed by the typewriter. Example:

```
ENTER PROGRAM
        1.  TYPE THIS PROGRAM COMPUTES THE INTEGRAL OF THE
            TYPE EXPONENTIAL OF X OVER THE RANGE FROM 0 TO 1 WITH
            TYPE AN INTERVAL SIZE OF 10.RS
        2.  TYPE WHEN THE COMPUTER PAUSES ENTER THE RANGE AND
            TYPE INTERVAL SIZE OF X.RS
```

For the card input of comments the user punches a C or a W in column 1 followed by a message beginning in column 7. The "C" card is ignored by the computer and can therefore be used to document a program. The "W" card causes a typewritten printing of its message and hence is similar to the TYPE instruction.

It should be noted that extensive comment statements use a lot of program core storage area. It is, therefore, a good idea to keep comments brief and concise.

XI. SENSE SWITCHES

XI. SENSE SWITCHES

There are four numbered program switches (or sense switches) on the IBM 1620 console. Using the Sampler, they can be employed to perform the following function:

Sense Switch 1	This switch can be employed as an indicator in an IF test. Example: IF SWITCH 1, THEN . . . . ., OTHERWISE . . . . . .RS. If Sense Switch 1 is in the "on" position, the THEN clause is executed, and if it is in the "off" position, the OTHERWISE phrase is executed.
Sense Switch 2	When typing a function register under the DISPLAY instruction, if this switch is "off", only the last number in the function register is displayed; when "on", all array values are displayed.
Sense Switch 3	If this switch is on during execution the program's source language steps are typed out.
Sense Switch 4	When this switch is turned on it interrupts the execution of a program and causes a branch to the typewriter for the users instruction.

These 4 sense switches affect only the typewriter and not the card reader. As for example,

```
CARD X .RS
```

gives all the values of the X array regardless of sense switch positioning. It should be noted that the normal position for these switches is off.

XII. PROGRAMS IN THE AMTRAN SAMPLER SUBROUTINE PACKAGE

XII. PROGRAMS IN THE AMTRAN SAMPLER SUBROUTINE PACKAGE

The console programmed subroutine package is a separate card deck from the Sampler and can be read in only after the Sampler is in the computer. To enter these subroutines into the system the instruction:

READ . RS

must be typed and a SUPPRESS and END card placed one at the front, and one at the end of the card deck of subroutines.

The following programs are contained in this package (they can all be read in as one deck or the user can pick out only those which he needs):

FORWARD	This program computes the forward difference of the expression following its mnemonic label.
RANGE OF	This subroutine is used to generate a function array over a specific range and interval size. After the mnemonic RANGE OF is typed a function register (S-Z) must be entered followed by a comma, after which the minimum and

maximum value and the number of intervals of the array are entered (each separated by a comma). Example:

RANGE OF X, 1, 2, 10.RS

where $1 \leq X \leq 2$, $\Delta X = .1$

DERIV	The DERIV subroutine computes the numerical derivative of the mathematical equation declared after its name. This subroutine calls the program FORWARD.
INT	This subroutine computes the definite integral of algebraic statements. If any of the function array registers (S-Z) are declared as the independent variable, the limits of integration are taken to be the range of the array variable. In other words, the independent variable must be defined in a RANGE OF instruction prior to its being used under the INT operand. This subroutine calls the program FORWARD.
LOAD	This subroutine loads the 10 numbers required for a 5 pt. Gaussian integration

into the W & V registers. It requires that the lower and upper limits of integration be given. Example:

```
1.  LOAD 1, 10 .RS
```

where 1 is the lower limit and 10 is the upper limit of integration.

GAUSSINT

This subroutine computes the definite integral of the expression which follows it, using a 5 pt. Gaussian integration scheme. To use GAUSSINT, the LOAD subroutine should have been used prior to it, since GAUSSINT uses the 10 numbers in the W & V registers as well as the upper and lower limits of integration specified in the LOAD program. To change the limits of integration without recalling the LOAD program, set the lower limit of integration equal to A and the upper limit equal to B. Example:

```
1.  3 = A.RS
2.  4 = B.RS
3.  GAUSSINT EXP X .RS
```

DEFINT

The DEFINT subroutine sets up conditions for a 5-pt. Gaussian integration scheme. It requires that the lower and upper limits of integration and the integrand be specified. Example:

```
DEFINT 0, 10, EXP X.RS
```

This subroutine calls both of the programs LOAD and GAUSSINT.

RUNGEONE

This subroutine employs a 4th order Runge-Kutta method for solving equations of the type:

$y' = f(x,y)$

This subroutine is called by typing

```
RUNGEONE RS.
```

The computer will then ask for the step size, XMAX, X (0), Y(0), and the function F(X,Y) to be integrated.

RUNGETHREE

This subroutine employs a 4th order Runge-Kutta method for solving equations of the type:

$y'' = f (x,y)$

This subroutine is called by typing RUNGETHREE RS. The computer will then ask for the step size, XMAX, X(0), Y'(0), and the function F(X,Y) to be integrated.

RUNGEFOUR

This subroutine employs a 4th order Runge-Kutta method for solving equations of the type

$$y'' = (x, y, y')$$

This subroutine is called by typing RUNGEFOUR RS. The computer will then ask for the step size, maximum value of X, X(0), Y(0), Y'(0) and the function F(X(N), Y(N), Z(N)) to be integrated.

TYPEOUT

This subroutine types the equation and results of the expression which follows its mnemonic label. Example:

```
TYPEOUT 2 + 2 .RS
```

The computer will then type:

```
2 + 2 = 4
```

SET

The subroutine SET allows data to be input into the computer in any mixed

	format. To use the subroutine, enter the name SET followed by the minimum value of the data points, a comma, the maximum value of the data points, a comma, the number of data points, a comma, and then the actual data points (each separated by a comma).
INTERPOLATE	This subroutine uses a fifth order divided difference formula to interpolate. The values in the Y register are taken to be the abscissa values and those in the Z array to be the ordinate or functional values for Y. Y and Z must have the same range. The numbers in the X register are taken to be the new abscissa values. A 5th order divided difference formula is used to interpolate Y to find the new ordinate or functional values for Z which correspond to the abscissa values for X.
TYPEDATA	This subroutine outputs data from the typewriter. Two data points are typed per line; the one on the left is the

	abscissa, the one on the right is the ordinate or functional value corresponding to the abscissa.
PUNCHDATA	The subroutine PUNCHDATA punches one register array value per card in a format that can be read back into the computer.

A complete set of trigonometric subroutines are available:

TAN	(tangent)
COT	(cotangent)
SEC	(secant)
CSC	(cosecant)
ARCSIN	($\sin^{-1}$)
ARCCOS	($\cos^{-1}$)
ARCCOT	($\cot^{-1}$)
ARCSEC	($\sec^{-1}$)
ARCCSC	($\csc^{-1}$)
TANH	(hyperbolic tangent)
COTH	(hyperbolic cotangent)
SECH	(hyperbolic secant)
CSCH	(hyperbolic cosecant)
ARCSINH	(inverse hyperbolic sine)
ARCCOSH	(inverse hyperbolic cosine)

ARCTANH	(inverse hyperbolic tangent)
ARCCOTH	(inverse hyperbolic cotangent)
ARCSECH	(inverse hyperbolic secant)
ARCCSCH	(inverse hyperbolic cosecant)

(These trigonometric subroutines are used in the same manner as the SIN and COS and give the same degree of accuracy).

All of these subroutines are used just as the mnemonic labels intrinsic to the system, and must conform to the same rules. If the mathematical expression to be operated on is more than one term, it must be enclosed in parentheses, otherwise, a single space should separate the operator and argument. Also, the last sentence of each subroutine explanation tells whether a subroutine contains any nested subroutines (or programs). If a subroutine does contain a nested program, this program must be read into the system prior to reading in the subroutine. If a sentence is not found under a subroutine explanation giving the nested programs, then it can be assumed that there are none.

XIII. ERROR MESSAGES

XIII. ERROR MESSAGES

When an incorrect procedure or a mathematically improper statement are entered, the computer will give a typed error message as a diagnostic. The following error messages are contained in the AMTRAN Sampler:

ERROR CODE	DESCRIPTION
THAT NAME HAS BEEN USED	The name of a console program has already been used by another program.
M. I.	The statement entered is mathematically improper.
REENTER STATEMENT	Enter a corrected statement.
UNDEFINED SYMBOL	An unknown has been used without having been defined.
NEARING OVERFLOW	There are only 100 positions left in the console program area and therefore after 100 mnemonic labels, letters, and/or numbers have been entered the

	console program storage area will be full. To continue past this point means destroying parts of the basic Sampler interpretive deck.
LEFT PARENTHESIS	Too many left parentheses
RIGHT PARENTHESIS	Too many right parentheses

When the above errors are encountered, the user has a variety of correction possibilities available to him. Some of the more common ones are:

<u>ERROR CODE</u>	<u>CORRECTION</u>
THAT NAME HAS BEEN USED	Delete the previous program by typing DELETE followed by the name of the program. (this procedure frees the program's mnemonic label). Or, by typing a $RS, which erases the statement it follows, a new instruction can be entered assigning a different program label.
M. I. REENTER STATEMENT	These two messages, appearing together, automatically

	delete the incorrect statement, and renumber for the corrected expression.
UNDEFINED SYMBOL	In this case a statement must be added to define the symbol. This can be done by deleting the line containing the undefined symbol (type $RS), and then defining the symbol. After this definition is completed followed by the expression which contained the undefined symbol a comma can be entered or a ".RS" can be entered in which case the computer will number a new line for the expression.
LEFT PARENTHESIS RIGHT PARENTHESIS	By typing a $RS the user can reenter the incorrect statement with the corrected parenthesis.
NEARING OVERFLOW	The user would check to see how close he is to the end of his program. If it is over 100

positions, he must delete enough
programs in core to give him the
needed space. (A position is
required for a mnemonic label,
a letter, or a number).

In making corrections, the user may, if he wishes, enter operations one at a time. For example:

```
INT EXP X. RS
```

can be entered as

```
INT RS EXP RS X RS. RS
```

Causing the computer to carry out the designated operations a step at a time. This mode of operation can be very useful in debugging a faulty statement for it examines the intermediate results at each step along the way. In this way, an error can be pinpointed, for each RS causes execution of the preceding operator or symbol. Consider the case where a statement has been entered with too many right parentheses.

1. X-(Y + EXP X)) - 64.3 .RS   The computer would type RIGHT PARENTHESIS.

The user would then type $RS, after which the computer would renumber for the corrected statement. If it is not apparent which parenthesis is causing trouble, the statement can be entered a step at a time.

1. X RS - RS (RS Y RS + RS EXP RS X RS) RS) R S

At this point the computer would type RIGHT PARENTHESES.

again telling the user that the symbol before the last RS is the incorrect parenthesis. He could then reenter his statement as

X - (Y + EXP X) - 64.3 .RS

or, if he wanted to check the rest of the statement, he could type @RS, which causes the computer to backspace and eliminate the last entry (in this case the parenthesis) and retype up to the backspace position, so that the statement would read:

1. X - (Y + EXP X)

From here he could continue

RS - RS 64.3 RS. RS

In addition to the AMTRAN Sampler error codes, the following error messages used in the IBM 1620 floating point subroutines are typed out whenever necessary -

07	Floating divide - attempt to divide using a number with a zero mantissa divisor.
08	Floating square root - attempt to find the square root of a negative number.
09	Floating sine or floating cosine - exponent greater than mantissa length.
10	Floating sine or floating cosine - exponent less than mantissa length, greater than 03.
13	Floating LN or floating log - zero mantissa.
14	Floating LN or floating log - negative.

XIV. SOLVING A NON-LINEAR DIFFERENTIAL EQUATION

XIV. SOLVING A NON-LINEAR DIFFERENTIAL EQUATION

THE SOLVE OPERATOR

AMTRAN possesses a general purpose Runge-Kutta subroutine package for solving differential equations. To use this operator, simply enter SOLVE RS (no period after SOLVE). The computer will they type out the instructions:

ENTER THE NUMBER, N, WHICH DESCRIBES YOUR DIFFERENTIAL EQUATION.

DY/DX = F(X,Y)	1
DY/DX = F(X,Y,Z), DZ/DX = G(X,Y,Z)	2
D^2Y/DX^2 = F(X,Y)	3
D^2Y/DX^2 = F(X,Y,DY/DX)	4

For example: To solve the differential equation

$$y'' = y' + y - e^x$$

enter N = 4.

(Note that when the computer asks you to enter F(X, Y, . . .), you must enter all variables with the subscript N and the DY/DX entries must be entered as Z SUB N)

```
STEP SIZE = .1,

X SUB 0 = 0,

XMAX = 1,

Y SUB 0 = 1,

DY/DX SUB 0 (Z SUB 0) = 1,

F (X SUB N, Y SUB N, Z SUB N) = Z SUB N + Y SUB N - EXP X SUB N.
```

The integration process may be terminated at any time by turning on Sense Switch 4. The results of this example should be exp x $(0 \leq X \leq 1)$.

```
2. DISPLAY RESULTS.

RESULTS SUB 10 = 2.71828
```

PICARD'S METHOD

Picard's method of solving integral equations by successive iterations is well-suited to on-line computer operations. Generally speaking, Picard's method works as long as the norm of the differential operator is not too large over the abscissa range of the trial function. A few iterations are generally sufficient to tell the user whether or not his solution is going to converge.

The differential equation must be rewritten as an integral equation (thereby incorporating the boundary conditions). A first approximation of Y is obtained by generating a function over a range and for some chosen interval size. The integration is then carried out using the assumed first approximation to Y. The computed function Y is then inserted into the integral equation in place of the previous approximation of Y and the integration carried out again to obtain a new approximation to Y. This iteration cycle is repeated until further integrations produce no changes in Y. After each iteration, the value computed for Y is displayed and in this way the user can determine if it converges. For example, consider the differential equation:

$Y'' = y^3 \exp(-y) - 2x^2$ over the range $.2 \leq x \leq 1$. To solve this equation, rewrite it as

$$y' = \int_{.2}^{x} (y^3 + y \exp(-6) - 2x^2)\, dx + C.$$

At $x = .2$, $\int_{.2}^{.2} (y^3 + y \exp(-y) - 2x^2)\, dx = 0$

by definition and $C = y'(.2)$.

Then $y = \int_{.2}^{x} \int_{.2}^{x'} (y^3 + y \exp(-y) - 2x''^2)\, dx''dx' + xy'(.2) + C.$

at $x = .2$, $y(.2) = C$

Therefore, in integral form,

$$y = \iint (y^3 + y \exp(-y) - 2x''^2)\, dx''dx' + xy'(.2) + y(.2)$$

To solve this with AMTRAN:

ENTER PROGRAM

```
    1.  .0025 = A, .005 = B. .RS
```

```
2.  RANGE Y, .2, 1, 50. RS

3.  REPEAT 10, INT INT (YYY + Y EXP (-Y) + 2XX) + AX + B = Y

4.  DISPLAY Y.
```

The computer then types

```
Y SUB 50 = .169438

Y SUB 50 = .16945287

Y SUB 50 = .16944028

Y SUB 50 = .1694401

Y SUB 50 = .1694401

Y SUB 50 = .1694401

Y SUB 50 = .1694401

Y SUB 50 = .1694401

Y SUB 50 = .1694401

Y SUB 50 = .1694401
```

In this example four iterations were sufficient to produce convergence and further iterations simply give the same results.

XV. CONCLUSION

In summary, the present AMTRAN Sampler system provides the user with an efficient and convenient method of entering equations into the IBM 1620 computer and building high level operators which are equivalent to the basic instruction set intrinsic in the system.

Modifications and improvements are being made in the Sampler system to extend its capabilities and simplify its use. Therefore, all suggestions or problems encountered by the user are important to the authors of this manual who invite comments on the system. A keyboard version of AMTRAN which takes advantage of special hardware for input and output is now available. Recent improvements in software including dynamic memory allocation, are also available with this system.

At present, research effort is being advanced to convert the AMTRAN system to the Burroughs 5500 and the IBM 1130 computers.

APPENDIX A

LOADING THE SOFTWARE

1) Set the Typewriter

   A) Left margin set at 10
   B) 1st tab set at 18
   C) 2nd tab set at 45
   D) Right margin set at 90

2) Clear Memory

   A) Press RESET and INSERT
   B) Type in 160001000000
   C) Set PARITY and I/O program switches to PROGRAM
   D) Press RELEASE and START
   E) If no PARITY or I/O lights, press INSTANT STOP, otherwise repeat 2A),B),D)

3) Set PROGRAM Switches

   A) PARITY and I/O to STOP
   B) OVERFLOW to PROGRAM
   C) PROGRAM switches 1 through 4 to OFF

4) Load Decks

   A) Place yellow deck, followed by white deck, into READ hopper
   B) Press INSTANT STOP and RESET
   C) Press LOAD on the hopper
   D) The computer will stop on MANUAL after the white deck is loaded
   E) Press START to load the yellow deck
   F) Upon completion of loading the yellow deck, the following heading will be typed and the program will wait for an entry

```
                    ENTER PROGRAM
                      1.
```

Please note that this procedure is given for an IBM MOD 1 computer.

APPENDIX B

MNEMONIC LABELS INTRINSIC IN THE AMTRAN SAMPLER

For a detailed explanation of the constants and operators intrinsic in the system, the reader is asked to consult the INDEX where the page numbers for these explanations can be obtained. Those not explained in detail in the manual:

ABS	LOG	SETREG
INTERVALS	RESET	SUM
LEFT	RESULTS	XMAX
IN	RIGHT	XMIN

are given below.

ABS	The absolute value of the designated function is taken. The complete forms are: ABS Y.RS ABS (X - Y/Z).RS
INTERVALS	The number, N, of intervals of a function is given by this operator. The actual number of ordinates stored in a function register is N + 1. If 50 = INTERVALS, an array of 51 numbers can be stored in the function register.

LEFT	The complete entry is of the form LEFT Y.RS The referenced function register is loaded into the variable accumulator and left-shifted. The number in position 1 is moved to position 0. The number in location 2 is moved to location 1, and so on. The right most position is filled by extrapolation using the four preceding points.
LN	This operator computes the natural logarithm (base e) of the expression it precedes. To illustrate: 1. LN 10.3.RS 2. LN (X + Y/6).RS
LOG	This operator computes the logarithm to the base 10 of the expression following it. Example: 1. LOG (.7 + X/Y).RS
RESET	This operator is designed to restore the computer to its original state at the time the AMTRAN interpretive deck was read in. It resets pertinent address

registers, counters, etc., without, however, modifying any data or console programs. It is generally used as an erase and reset operation for everything except data and console programs.

RESULTS	This mnemonic is used to represent that register, known as the floating accumulator which moves up and down the stack of working registers temporarily storing intermediate and final results.

Consider the mathematical expression:

1. X + Y.RS

When a register is not declared to which the numerical result of an equation is to be transferred, the result is put in RESULTS (the floating accumulator). The accumulator acts as a temporary function and can therefore be addressed as a function register.

To illustrate:

1. X + Y.RS
2. SIN RESULTS + 1 = Z.RS

Statement 1 adds Y to X and puts the result in the accumulator which initially is Reg. 1. Statement 2 takes the sine of REG 1 (the accumulator containing X + Y) adds one to it, and transfers the result to the Z register. Because the accumulator moves after each storage of a number or array, a result in the accumulator must be called immediately after it is executed, otherwise the accumulator will change its location in the stack of working registers and the desired numerical result destroyed. For example reconsider statement 2 as

2. 1 + SIN RESULTS = Z.RS

In this instance the number 1 is loaded into the accumulator whipping out X + Y; therefore, 1 + SIN RESULTS $\neq$ SIN RESULTS + 1 $\neq$ Z. It must also be remembered that only one operation can be performed on the accumulator without causing it to change registers. For example: Suppose that after 1. X + Y.RS has been executed the user wants to integrate X + Y and add to it the sine of X + Y. (i.e. INT (X + Y) + SIN (X + Y) = RS. The instruction INT RESULTS + SIN RESULTS does not give the desired result.

If a hard copy of statement 1 is required, it can be obtained with either of the following instructions:

2. CARD RESULTS .RS - gives output on cards

2. DISPLAY RESULTS .RS - gives typed output

Since the two operators CARD and DISPLAY do not load into the accumulator, they cannot effect its contents.

To illustrate: CARD RESULTS can be nested safely between statements

1. X + Y.RS
2. CARD RESULTS .RS
3. SIN RESULTS + 1.RS

The purpose of not declaring a register to which the result of a mathematical expression are to be transferred, is to conserve storage registers. When a result is not needed for use later in a program, there is no reason for assigning it a storage register.

RIGHT	Similar to left shift, with the exception that the numbers in the specified function register are moved one position to the right. The left most position is filled by extrapolating the first 4 values.
SETREG	Sets the length of the working register. The instruction is: `SETREG N.RS` Where N is the number of intervals the user requires for his function arrays. The computer then determines the number of working registers available and types: `YOU NOW HAVE ______FUNCTION REGISTERS` The instruction SETREG N sets every function register to the same size. For N intervals the computer allow N + 5 number locations in each register; 1 for the constant accumulator, 1 for the minimum and 1 for the maximum value of the function, 1 for the interval size, and N + 1 locations for the array. Note that N intervals generates N + 1 numbers. This operator allows the user to tailor core to the size of his program and make

available to himself a larger number of working registers. Initially the number locations available to arrays are divided into 13 registers. In the 40 K version of AMTRAN each register is capable of holding an array of 51 numbers, and in the 60 K version, arrays of 101 numbers can be stored in each function register.

SUM — This operator replaces a function with the running summation of its ordinates. For example:

```
SUM X.RS
SUM (Y/X - 3).RS
```

The referenced function is loaded into the floating accumulator. The data in position 2 is then added to the data in position 1 and the result stored in location 2. This process continues until the data in position N is added to the data in location N-1 and the result stored in location N.

XMAX — Designates the upper limit of a function. It corresponds to X SUB (-2).

XMIN	Used to represent the lower limit to a function. It can also be designated as X SUB (-3).

APPENDIX C

In the Sampler there exist variable length function registers which can be addressed by numbers; REG1, REG 2, ... etc. or the alphabetic letters S, T, ... Z. These registers occur consecutively in core, that is REG 1, REG 2, ... REG N.

A total of 1050 numbers are allocated to the function registers. Initially the number of registers is set at 13, allowing 100 intervals (60K version) or 50 intervals (40K version) for each register. The last eight registers correspond to and can be addressed by the letters S, T, U, V, X, W, Y, Z.

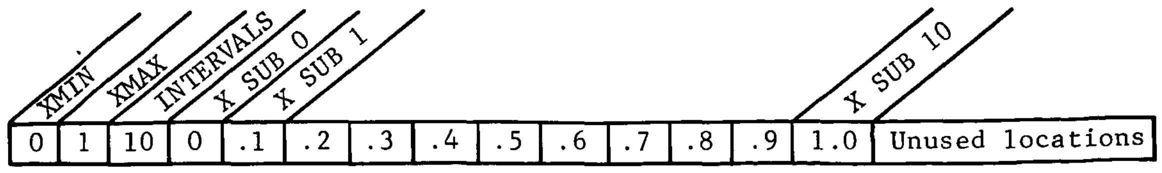
Each function carries with it the maximum and minimum values of its independent variable and the number of intervals used in its representation (N intervals generates N + 1 numbers). These three quantities are stored in special number locations set aside in each function register.

XMIN	XMAX	INTERVAL NUMBER	50 intervals (51 numbers) or 100 intervals (101 Numbers)

Each block represents one number location which will hold an eight digit number. Example:

1. RANGE OF X, 0, 1, 10. RS

This instruction generates X over the range from 0 to 1 with 10 intervals, and then automatically stores the result in the indicated register (X in this example). The X register would have the following appearance:



10 intervals, 11 numbers

Also,

XMIN = XSUB(-3)

XMAX = XSUB(-2)

INTERVALS = XSUB(-1).

XSUB(-4) is a special location called the constant accumulator.

If a constant is set equal to a function register, say 5 = X, then the 5 is stored in the first location of the designated register (XSUB-4). Transferring a constant into a function register causes the interval number of the function register to be set equal to zero.

To illustrate:

SETREG K.RS,

where K is the interval size, automatically sets each register to K + 5 locations. The computer then calculates the number "J" of registers available with this interval size and types:

YOU NOW HAVE J FUNCTION REGISTERS

An interval size of 50 gives 19 function registers, 8 storage and 11 working. With this many working registers it is almost impossible to overflow into the storage areas S, T, ... Z.

In order to interpret mathematical statements correctly the AMTRAN system operates with a variable or floating accumulator (mnemonic code RESULTS) which moves up and down the stack of working registers. The accumulator is not, however, a separate register and at the beginning of each program it is REG 1. Consider the following series of mathematical statements:

1. RANGE DF X, 2, 3, 10.RS
2. RANGE OF Y, 1, 2, 10.RS
3. RANGE OF T, 7.5, 8.5, 10.RS
4. XY + T(X-Y) = W.RS

The first three statements would cause the working and storage registers to look as they do in Figure 1.

storage register	REG 10 or Z															
	REG 9 or Y	1	2	10	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2.0	(unused space)
	REG 8 or W															
	REG 7 or X	2	3	10	2.0	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9	3.0	(unused space)
	REG 6 or V															
	REG 5 or U															
	REG 4 or T	7.5	8.5	10	7.5	7.6	7.7	7.8	7.9	8.0	8.1	8.2	8.3	8.4	8.5	(unused space)
	REG 3 or S															
working registers	REG 2															
	REG 1															

Statement 4. XY + T(X-Y) = W.RS causes the following position changes of the accumulator:

a. Contents of X register are loaded into the accumulator (REG 1).

b. Accumulator moves up to REG 2 where Y is then loaded.

c. When the computer encounters the "+" sign, X is multiplied by Y; the accumulator moves back to REG 1 and stores this product.

d. Accumulator moves to REG 2 and stores T.

e. Finding the parenthesis the accumulator moves to REG 3 or the S register and loads X. NOTE: Working registers and storage registers can overlap.

f. The accumulator moves up to REG 4 or the T register and stores Y. NOTE: The contents of the T register have been destroyed permanently and replaced with the Y register's values. If T is called again it will give Y.

g. Encountering the right parenthesis the computer subtracts Y from X; the accumulator moves down the stack to REG 3 and stores this result.

h. T is then multiplied by REG 3 and the result stored in REG 2.

i. Finding the equal sign, the computer adds REG 2 (containing T(X-Y)) to REG 1 (containing XY) and the result is stored in REG 1.

j. The computer loads REG 1 into the W register.

In the preceding example large blocks of each function register were unused. Also, the original array in the T register was destroyed because the accumulator worked its way into the stack of storage registers. To avoid this occurrence a longer stack of working registers could be created. If it is known that the maximum interval size of the array's in a program is $K < 50$ in the 40K version (and $K < 100$ in the 60K version), the user can set the number of registers to optimize core.

REFERENCES

1. M. R. Albert; P. L. Clem, Jr.; V. A. Dauro; L. Morrow, Jr.; Northrop Space Laboratories, Huntsville, Alabama; and J. Reinfelds; A. J. Scott; R. N. Seitz; L. N. Wood; NASA, MSFC, Huntsville, Alabama, "The AMTRAN Sampler System," September 1965.

2. Robert N. Seitz, "AMTRAN, An On-Line Keyboard Computer System for Scientific and Engineering Use," NASA TM X-53243, May 4, 1965.

3. J. Reinfelds; L. A. Flenker; R. N. Seitz; P. L. Clem, Jr.; "AMTRAN, A Remote - Terminal, Conversational - Mode Computer System," March 1966.

INDEX

JULY 1966 APPROVAL TM X-53342

THE AMTRAN SAMPLER SYSTEM
INSTRUCTION MANUAL (REVISED)

By

Albert, M. R.; Clem, P. C.; Flenker, L. A.;
Reinfelds, J.; Seitz, R. N.; and Wood, L. H.

This information in this instruction manual has been reviewed for security classification. Review of any information concerning the Department of Defense or Atomic Energy Commission programs has been made by the MSFC Security Classification Office. This report, in its entirety, has been determined to be unclassified.

This document has also been reviewed and approved for technical accuracy.

Ernst Stuhlinger

ERNST STUHLINGER
DIRECTOR, RESEARCH PROJECTS LABORATORY

DISTRIBUTION

DIR
Dr. von Braun

R-DIR
Dr. McCall

R-RP
Dr. Stuhlinger
Mr. Heller
Mr. Bucher
Mr. Wood
Dr. Shelton
Dr. Seitz (150)

R-AERO
Dr. Geissler

R-ASTR
Dr. Haeussermann

MS-IL (8)
MS-T
Roy Bland (100)

R-COMP
Dr. Hoelzer
Mr. Bradshaw
Mr. Prince
Mr. Lynn
Dr. Krenn
Mr. Felder
Mr. Joseph

R-P&VE
Mr. Cline

AST-S
Dr. Lange

R-AS
Mr. Williams

MS-IP

CC-P
MS-H

Mr. East
OART-REI
National Aeronautics and Space Administration
Washington, D.C. 20546

Computation & Data Reduction Center
TRW Space Technology Laboratory
One Space Park
Redondo Beach, California
Dr. Burton D. Fried
Dr. Eldred Nelson
Mr. William Sassaman

Dr. Ivan E. Sutherland
Room D-200
Advanced Research Projects Agency
The Pentagon
Washington, D.C.

Mr. Gary Clark
Vice President, Marketing
Defense and Space and Special Systems
Burroughs Corporation
Paoli, Pennsylvania

Prof. T. Oettinger
Computation Laboratory
Harvard University
Cambridge, Massachusetts

Dr. Glen J. Culler
Computation Laboratory
University of California
Santa Barbara, California

OART-RRA
National Aeronautics and Space Administration
F.O.B. 10
Washington, D.C. 20546
Dr. Gary Etgen
Dr. Raymond Wilson
Dr. Hermann Kurzweg

Mr. Jack Ault
Defense and Space and Special Systems
Burroughs Corporation
Paoli, Pennsylvania

Scientific and Technical Information Facility
Attn: NASA Rep S-AK/RKT   (25)
P.O. Box 33
College Park,
Maryland